Amendments to the Specification:

Please replace the paragraph beginning on page 4, line 8 with the following amended paragraph:

To overcome the above shortcomings of the prior art of macro language processors, the present invention provides an extensible macro language that allows users to write new macro commands that include procedures tailored to the specific needs of the users without a need to modify any source code of the macro language processor. The extensible macro language is enabled to process the new macro commands by recognizing the new macro commands unknown to the language and associating the new macro commands with procedure calls stored in a registry, i.e., a repository, thereby allowing dynamic extension of a macro language.

Please replace the paragraph beginning on page 10, line 14 with the following amended paragraph:

The syntax or the grammar employed in one embodiment of the extensible macro language will now be described in detail. The extensible macro language of the present invention includes a syntax ~~(Figure 1-104)~~ (Figure 1 (104)) comprising literals, macros, comments and operator/scoping characters.

Please replace the paragraph beginning on page 11, line 5 with the following amended paragraph:

Macros include instructions to the macro processor, like procedures or functions in a programming language. According to the syntax defined in the present invention, all macros are embedded within curly braces. In one embodiment, the macros ~~fall into two categories: procedures macros and iterator macros~~ may be categorized as procedure macros and/or iterator macros.

Please replace the paragraph beginning on page 11, line 12 with the following amended paragraph:

Procedure macros are designed to perform some work. They may expand to a value[[;]], they may declare a variable[[;]], and/or they may invoke a process. The actions performed are entirely specified by the designer of the macro. In one embodiment, the macros must, however, return a "true" value upon successful completion of their task and a "false" value upon failure.

Please replace the paragraph beginning on page 14, line 1 with the following amended paragraph:

The conditional blocks have internal scope, i.e., the failure of a conditional block does not affect the surrounding code. For conditions in a block to affect the outer block, the syntax additionally includes what is referred to as a propagating conditional denoted by angle brackets. If any macro within a pair of angle brackets fails, the block within the angle brackets as well as the next outer block fails. The following examples illustrate a macro expression with a conditional and propagating conditional:

Please replace the paragraph beginning on page 15, line 4 with the following amended paragraph:

Figure 2 illustrates an example of a macro expression including an iterator macro of the present invention. As described with reference to Figure 1, the keyword "ForEach" is recognized by the parser 102 (Figure 1) as a macro, and the word "Employee" is recognized as a parameter to the macro "ForEach". When the macro handler receives the token keyword "ForEach", the macro handler 110 (Figure 1) performs a look-up of the keyword "ForEach" in the registry 112 and executes the corresponding code. The code for "ForEach" macro, for example, may include instructions to perform commands found within the begin/end block of the macro expression for all sub-objects 204b[[,]] and 204c in a given object 204 having the type of the specified parameter "employee". In this macro expression 202, another macro exists within the begin/end block. Accordingly, the macro handler 110 (Figure 1) performs a look-up of the keyword "Property" in the registry 112 and executes the corresponding code for each of the sub-objects 204b[[,]] and 204c having employee type as specified- in the "ForEach" keyword. The code associated with the "Property" keyword, for example, may include instructions to print the value of the type specified in the parameter of the keyword "Property", in this case, an employee name as specified by "EmpName". Consequently, the result of the macro expression 202 is the output shown at 2 08, "Mary John".